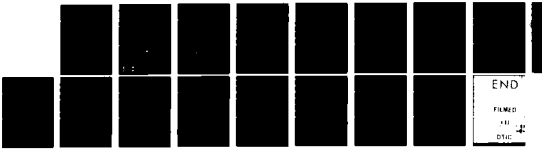
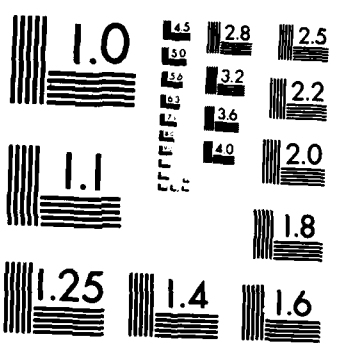
AD-A123 588 TOP-DOWN CELLULAR PYRAMIDS(U) MARYLAND UNIV COLLEGE PARK COMPUTER VISION LAB A Y WU ET AL. JUL 82 TR-1191 AFOSR-TR-83-0865 AFOSR-77-3271 1/1

UNCLASSIFIED F/G 9/2 NL

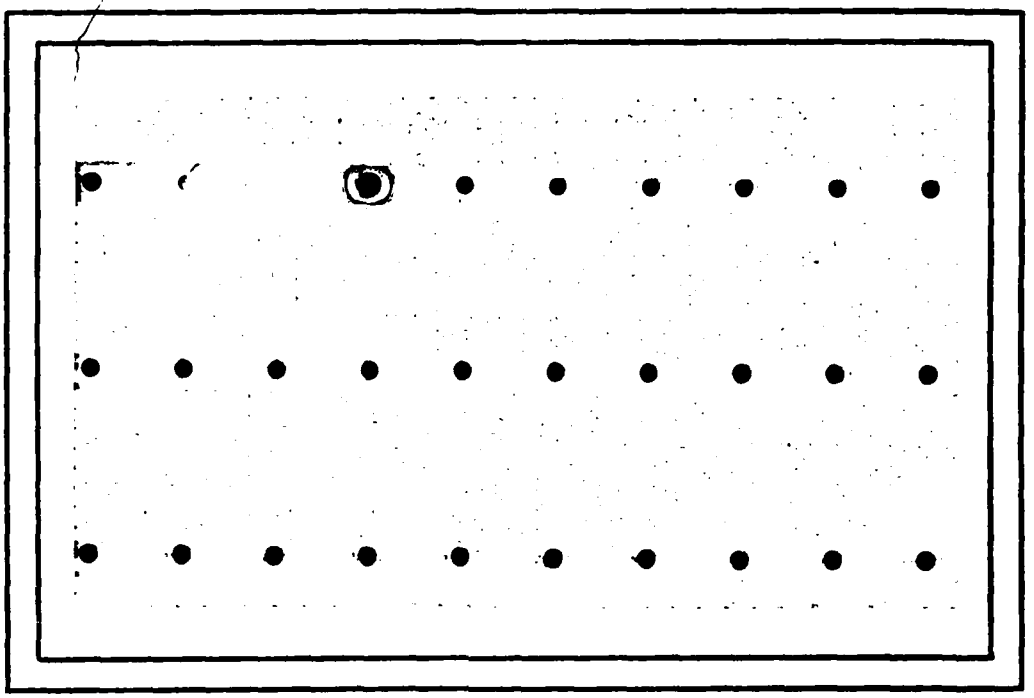




AFOSR-TR- 83-0065

(4)

AD A125588



COMPUTER SCIENCE TECHNICAL REPORT SERIES

Approved for public release; distribution unlimited.

DTIC FILE COPY

UNIVERSITY OF MARYLAND
COLLEGE PARK, MARYLAND
20742

88 03 14 08[illegible]

UNCLASS[illegible]

B

TR-1191
AFOSR-77-3271

July 1982

TOP-DOWN CELLULAR PYRAMIDS

Angela Y. Wu*
Azriel Rosenfeld

Computer Vision Laboratory
Computer Science Center
University of Maryland
College Park, MD 20742

ABSTRACT

A cellular pyramid is an exponentially tapering stack of arrays of processors ("cells"), where each cell is connected to its neighbors ("siblings") on its own level, to a "parent" on the level above, and to its "children" on the level below. It is shown that in some situations, if information flows top-down only, from fathers to sons, then a cellular pyramid may be no faster than a one-level cellular array; but it may be possible to use simpler cells in the pyramid case.

The support of the U.S. Air Force Office of Scientific Research under Grant AFOSR-77-3271 is gratefully acknowledged, as is the help of Janet Salzman in preparing this paper.

*Also with the Department of Mathematics, Statistics, and Computer Science, the American University, Washington, DC 20016.

1. Introduction

There has been much research on parallel processors because they can provide significant speedups for many tasks. Many multiprocessor machines and VLSI architectures have been designed and studied, and some have actually been built or are under construction [1-9]. Many parallel algorithms have also been studied. Image processing and graphics are potentially important areas of application of parallel processing because of the size and complexity of the data (pictures) to be processed or generated. On special purpose processors and general VLSI architectures for raster graphics and image processing see [7-12].

The most natural and simple multiprocessor architecture for image processing and raster graphics is a two-dimensional array of processors where a processor is assigned to a pixel (or a rectangular block of pixels), with neighboring pixels (blocks) represented by neighboring processors which can communicate with each other [13]. ILLIAC, MPP, and CLIP are examples of cellular array machines [1-3].

An extension of the concept of a cellular array is a hierarchical structure of processors called a pyramid. Variations on the pyramid concept include recognition cones and quadtree machines [14-17]. A basic pyramid machine consists of a series of levels, each of which is a two-dimensional cellular array, where the numbers of processors at successive levels decrease

by one-half in each dimension. The processors at higher levels represent larger blocks of pixels (four times the number of pixels represented by the processors on the level immediately below), with the highest level having only one processor called the APEX or ROOT representing the entire picture. A processor is linked to processors representing neighboring blocks at its own level. It is also linked to the processor (called its parent), at the level above it, representing a bigger block of which it is a subblock, and it is linked to the four processors (children) at the level below it, representing each of its four subblocks.

A quadtree machine can be considered as a special case of a pyramid where processors at the same level are not linked to each other. The basic non-overlapping pyramid just described has also been generalized to the so-called overlapped pyramids where each processor can have up to four parents, and to pyramids where each processor's neighborhood can be changed dynamically [18]. Pyramids to perform strictly bottom-up computation for image analysis, i.e., allowing only upward communication, have been investigated [19,20]. In this paper, we study the performance of pyramids and quadtrees which allows information to flow downward only.

Section 2 gives a more precise definition of a top-down pyramid. Section 3 contains examples of the use of top-down pyramids. The performance of top-down pyramids, in particular, when compared to cellular arrays, is discussed in Section 4.

2. Top-down pyramids and quadtrees

Let $P(x,y)$ be a 2^n by 2^n picture for some $n>0$, with pixels having coordinates $0\le x\le 2^n-1$ and $0\le y\le 2^n-1$. A pyramid is an $(n+1)$-level structure of square arrays of processors (sometimes called cells). Level 0 is the base array containing $2^n\times 2^n$ cells. Level n contains only one cell called the ROOT or APEX cell. In general level k $(0\le k\le n)$ contains $2^{n-k}\times 2^{n-k}$ cells. Hence there are a total of $(2^{2n+2}-1)/3$ cells, less than one-third more than in a cellular array. Let (i,j,k) specify the processor (cell) at level k with coordinates (i,j) in the k-th level array, $0\le k\le n$, $0\le i\le 2^{n-k}-1$, $0\le j\le 2^{n-k}-1$. Then (i,j,k) is assigned the block of $2^k\times 2^k$ pixels in $P(x,y)$ such that $i2^k\le x<(i+1)2^k$ and $j2^k\le y<(j+1)2^k$. Cell (i,j,k) is connected to its four siblings $(i-1,j,k)$, $(i+1,j,k)$, $(i,j-1,k)$ and $(i,j+1,k)$. If $i,j=0$ or $2^{n-k}-1$, the border processors have two or three siblings only. Processor (i,j,k) is also connected to its parent processor $(\lfloor\frac{i}{2}\rfloor,\lfloor\frac{j}{2}\rfloor,k+1)$ for $0\le k<n$, and to its four children $(2i,2j,k-1)$, $(2i,2j+1,k-1)$, $(2i+1,2j,k-1)$, $(2i+1,2j+1,k-1)$ for $0<k\le n$. In general, in a pyramid, the processors are identical, the data flow at each communication link is bidirectional, and the width of the data paths is assumed to be the same so that the communication time is identical for all processors.

In a top-down pyramid, the data flow to siblings is bidirectional, but data can only flow from parent to children. In a strictly top-down pyramid, there is no data flow among siblings. Therefore, a strictly top-down pyramid is equivalent to a top-down quadtree. If a processor M needs information from its

sibling N, because of the lack of communication between siblings, M would have to simulate N to calculate the result provided the parent also gives M the instructions and data given to N.

One motivation for studying top-down pyramids comes from their potential usage in generating graphics images. The description of the images to be generated can be input to the APEX. Data and instructions can be transmitted downward to the bottom level processors which can be connected directly to and thus control the display (which we may assume to be a mosaic-type panel display in which each element is directly driven by a bottom-level processor). The processor at the top level can divide the image to be generated into subimages for each of its children whose job should thus be simpler and can be further divided.

3. Example: line generation

Consider the task of generating a digital straight line joining two given points A,B whose coordinates (x_a,y_a), (x_b, y_b) are given to the APEX cell of a top-down pyramid. We assume a $2^n \times 2^n$ picture, as described in Section 2. The APEX can find the equation (slope and intercept) of the line $\overline{AB}$. It can then find the intersection of the line $\overline{AB}$ and the interior border lines $x=2^{n-1}$, $x=2^{n-1}-1$, $y=2^{n-1}$, $y=2^{n-1}-1$, which divide the picture into four quadrants. These intersections together with A,B become endpoints of (possibly) shorter line segments contained in the APEX's children blocks. Note that at most three of the children will contain parts of line $\overline{AB}$. The equation of $\overline{AB}$ and the appropriate endpoints will then be transmitted to the appropriate processors at level n-1. The above procedure for finding intersections is repeated at each level until the bottom level is reached. Since each process at level 0 represents only one pixel, no further subdivision is necessary and the pixel can decide if it is on the line $\overline{AB}$.

Let t_1 be the time needed to find the equation of $\overline{AB}$, t_2 the time to find the intersection of $\overline{AB}$ (knowing its equation) and an interior quadrant border, and t_3 the time for a processor to transmit a datum (say an x or y coordinate) to its children. Then regardless of the length of $\overline{AB}$, it takes $t_1 + (4t_2+6t_3)n$ units of time to generate line $\overline{AB}$.

One important point that is worth noting in the above algorithm is that the intersection of line $\overline{AB}$ and a quadrant boundary may not be an integer. Rounding or truncation gives very inaccurate line segments if n is large. Hence the number of significant digits in the fraction increases as the lower levels of the pyramid are reached. One question that arises is whether the number of significant digits will ever become too large. If we consider the coordinates to be in binary, then the number of significant bits after the binary point increases by 1 as we go down the pyramid by one level. But each dimension of the block represented by the processor decreases by one-half and thus the number of bits needed to address the pixels in the block decreases by 1. Hence the total number of bits needed remains constant and the level the processor is in tells the location of the binary point.

4. Comparisons

In Section 3 we showed that a top-down pyramid can generate a digital straight line joining two given endpoints $A=(x_a,y_a)$, $B=(x_b,y_b)$ in time proportional to the height of the pyramid which is n for a $2^n \times 2^n$ picture. In the algorithm, no interaction between sibling processors was needed; therefore a strictly-top-down pyramid (top-down quadtree) gives the same result. In sequential raster graphics, a straight line is often generated by loading the frame buffer using some incremental point plotting algorithm, and the time required is proportional to $\max(|x_a-x_b|,|y_a-y_b|)$ which can be as large as 2^n [21,22].

Consider a single instruction stream multiple data stream (SIMD) machine M where the processors are interconnected as a two-dimensional cellular array. M has a control unit which broadcasts instructions (or data) to all the processors in the array. Assuming that each processor represents one pixel, M can generate a digital straight line as follows: the controller broadcasts the coordinates of the two endpoints A,B and the instructions for each processor to check whether the pixel it represents lies on the line $\overline{AB}$. Then each processor can set its pixel values accordingly. This can be done in t_4+2t_5 units of time where t_4 is the time needed to check if a pixel lies on $\overline{AB}$ provided the processor knows the coordinates of the pixel in the picture. Note that $t_4<t_2$. t_5 is the time needed to broadcast a coordinate to all the processors. In

general, t_5 is assumed to be a constant since the broadcasting takes place at signal transmission speeds. This is reasonable when the number of processors involved is modest. However, if the number of processors is large, say on the order of a quarter of a million for a 512 by 512 picture, direct (hardwired) broadcasting requires too much power. Thus some fanout process is needed. This means there is implicitly a tree (or pyamidlike) structure in which at each node (junction) there is some kind of amplifier, power booster or repeater instead of a processor. The height of the tree is $\log_b$ (number of processors) where b is the fanout factor. Hence $t_5 = t \cdot \log_b$(number of processors) where t is the propagation delay time at each node. (In terms of TTL logic, t is approximately 3 to 4 nanoseconds, but a 4-bit add operation takes approximately 12 to 15 nanoseconds and reading into a register takes approximately 10 to 12 nanoseconds [23].) Therefore, strictly speaking, the broadcasting time t is proportional to log (number of processors) with a small constant of proportionality. But since almost all computations involve much more than a simple addition, while t_5 is much faster, it is not unreasonable to consider t_5 as a constant. Under this assumption, a straight line can be generated in a small constant amount of time regardless of the size of the picture or the length of the line.

The straight line generation example illustrates an important property of strictly top-down pyramids (top-down quadtrees). The information (or instruction) that starts at the APEX cell takes

at least n (height of the pyramid) steps to get to the bottom level cells. The instruction a level k cell (i,j,k) receives from its level k+1 parent $(\lfloor\frac{i}{2}\rfloor,\lfloor\frac{j}{2}\rfloor,k+1)$, and the instructions (i,j,k) sends to its level k-1 children, may be different. However, based on the top-down nature of the pyramid, these differences are only results of the information from $(\lfloor\frac{i}{2}\rfloor \quad \frac{i}{2}\rfloor,$ k+1) and the knowledge of which quadrant each child is in which is nothing more than its coordinates. Therefore, w a message from the APEX cell reaches the bottom level of a top-down pyramid, the only additional information the message can possibly contain is the (global) coordinates of the processor. Thus if each cell at the bottom level knows its coordinates and it knows the input to the APEX, each cell can simulate its level n ancestor at the first step, then simulate its level n-1 ancestor at the second step. Continuing in this way, at the end of n steps, the results at the bottom array are the same as if the information had trickled down the pyramid level by level. Thus there is no need for any of the intermediate level processors, i.e., one third fewer processors can be used and what we have is a two-dimenisonal SIMD array machine. Because there is no inter-level processor communication, the same task can be performed by the array machine in time no more than that needed by a strictly top-down pyramid. In many cases some of the work, such as the quadrant border intersection finding in the digital straight line generation example, done by the level 1 to level n-1 processors, is no longer necessary.

Conversely, a top-down pyramid can simulate a cellular array by having all the non-bottom level processors act as message transmitters only. As soon as a processor receives a message, it forwards the message to all its children without any processing. Then the time it takes to generate a straight line is t_4+2t_6 where t_6 is the time it takes a message from the APEX to reach the bottom level. So $t_6=nt$ where t is the time for a processor to forward a message. In terms of TTL logic, t is approximately 20 nanoseconds. This is not much slower than a cellular array and the intermediate level processors do not do any computation.

The above discussion shows that every task which can be performed by a strictly top-down pyramid can be done by a cellular array at the same or faster speed, when the processors in both systems have the same processing power and each processor in the array knows its coordinates in the picture. (Note that in a pyramid, each processor only needs to know which child it is of its parent, but not its global coordinates.) In the above, we also implicitly assume that the processors can store all the information from the controller or the APEX before the computation starts, or at least the processors can accept all the information without having to wait for the completion of computations, or the computations are so simple that they are always completed before further information arrives. In the rest of this section, we will show that a top-down pyramid can be faster if

these implicit assumptions are not satisfied, because a top-down pyramid has the advantages of a pipeline machine architecture when information is fed to the APEX processor to be transmitted down continuously.

First, consider the simple example of generating many digital straight lines where the APEX is given the sequence of pairs of endpoints. Clearly, the straight lines can be generated one at a time. As soon as the level i cell has transmitted the proper information to its level i-1 children, it can accept the information for the next line from its parent at level i+1. Thus the total time it takes a pyramid to generate m lines in an $2^n \times 2^n$ image is less than $2(t_1+(4t_2+6t_3)n)+m-1$ which is proportional to n+m. When the endpoints are broadcast by the controller, a cellular array can generate the same m lines in time $(2t_3+t_4)\cdot m$. The orders of the two time complexities are about the same when $m \geq n$, but for large m, the pyramid is faster.

Next, consider another example in which the pixel values of a picture to be generated depend on the results of a certain computation which is a function f of the binary representation $(x_0x_1\ldots x_n, y_0y_1\ldots y_n)$ of the coordinates (x,y) of the pixels and some input value z. The computation of f(x,y,z) is a sequence of calculations $g_0,g_1,\ldots,g_n$ such that g_0 uses only the values of x_0,y_0 and z, g_1 uses only x_1,y_1,z and the result of g_0, $\ldots,g_i$ uses x_i,y_i,z and the result of $g_i-1,\ldots$;f(x,y,z) is the

result of g_n. For instance, input z may be the coordinates of some points, and the task is to perform certain transformations of the coordinates of the pixels are within a certain range of z. If there are m input values $z_1, z_2, \ldots, z_m$ then a cellular array can generate the picture in O(mn) time since it has to finish processing one value before it can start on the next one. If a top-down pyramid structure is used, the m input values can be fed into the APEX continuously. While the level i cells are calculating $g_{n-i}(z_j)$, the level i+1 cells are calculating $g_{n-i-1}(z_{j+1})$. Thus, the picture can be generated in O(n+m) time. This is almost an ideal situation where the pipeline feature of top-down pyramids is being used to its fullest advantage, because the entire computation can be divided so that the processors at each level do one n^{th} of the work and the partial results are needed by the processors on the next level.

When a pyramid is used for image processing or raster graphics, if each processor represents only one pixel, the number of processors is very large. A 512×512 image needs one quarter (or one third) of a million processors for a cellular array (or a pyramid). Therefore it is a good idea to use processors which are as simple as possible. A pyramid structure has the possible advantage that only simple processors are needed since only parts of the tasks (hopefully simpler than the entire task) need be performed at each level. For example, consider the simple coordinate matching problem using very simple processors which can compare only one

bit at a time. A SIMD cellular array takes c_1nm time to match m n-bit numbers, whereas a pyramid takes c_2n+m time. Moreover, in the cellular array, every processor is doing the matching process all the time, whereas in a pyramid, many of the processors can be doing something else.

In the above, we have considered strictly top-down pyramids. The power of a (not strictly) top-down pyramid, where sibling communication is allowed, is almost the same as that of a strictly top-down pyramid. A strictly top-down pyramid can simulate the top-down pyramid in at most four times the computation time, since all the information obtained from the siblings comes from computation on information from the parent cell and the coordinates of the siblings, which are known and thus can be simulated one by one.

5. Concluding remarks

For certain tasks, if we assume that a controller can broadcast to all the processors in a small constant amount of time, then a top-down pyramid is not always faster than a cellular array. It is not much slower than a cellular array either because it can always have the bottom level act as a cellular array and the other processors simply forward the instructions downward without using their processing power. However, when a task can be divided up into subtasks for various levels' processors to perform, the pipeline architecture features of a top-down pyramid can be used effectively and great speed-up over cellular arrays or sequential processing can be achieved.

Another advantage of pyramids is that the processors used can be simpler than those in a cellular array since the entire task is divided up into subtasks which should in general be easier to perform. The processors do not need to know the global coordinates of the pixels; thus the lower level processors need to process fewer bits. This is significant if the number of processors involved is very large. One possibility for further study is to have processors of different powers at different levels. If a bottom level processor is responsible for only one pixel, it can be the simplest.

Unrestricted pyramids which allow information to flow both upward and downward are potentially useful for raster graphics, too. In particular, in interactive graphics the description of a picture

to be generated can be input at the APEX. When an interactive input device points to some pixels or draws some curves, the input at the bottom level of the pyramid can be transmitted upward (and sideways) to be processed. An upper level cell which sees a bigger block of the image can also send information down to its children based not only on information from its parent and siblings but also from the state of the part of the picture it oversees. Using pyramids for interactive raster graphics deserves further study.

References

1. K. E. Batcher, "Design of a massively parallel processor," IEEE Trans. Computers 29, 1980, 836-840.

2. M. J. B. Duff, "CLIP4: A large scale integrated circuit array parallel processor," Proc. Third IJCPR, 1976, 728-733.

3. W. Bouknight, et al., "The ILLIAC IV System," Proc. IEEE 60, 1972, 369-388.

4. C. Rieger, "ZMOB: Doing it in parallel," Proc. Workshop on computer architectures for pattern analysis and image database management, 1981, 119-124.

5. A. P. Reeves, "A systematically designed binary array processor," IEEE Trans. Computers 29, 1980, 278-287.

6. D. Kuck, "A survey of parallel machine organization and programming," ACM Computing Surveys 9, 1977, 29-59.

7. M. J. B. Duff and S. Levialdi (eds.), "Languages and architectures for image processing," Academic Press, 1981.

8. K. Preston, Jr. and L. Uhr, "Multicomputers and image processing," Academic Press, 1982.

9. Proc. Workshop on computer architectures for pattern analysis and image database management, 1981.

10. S. Gupta, R. F. Sproull, and I. E. Sutherland, "A VLSI design for updating raster-scan displays," Computer Graphics 15, 1981, 71-78.

11. J. H. Clark, "A VLSI geometry processor for graphics," Computer 13 (7), 1980, 59-68.

12. H. Fuchs and J. Paulton, "Pixel-planes: a VLSI-oriented design for a raster graphics engine," Computer Graphics 15, 1981, 80-81.

13. S. H. Unger, "A computer oriented toward spatial problems," Proc. IRE 46, 1958, 1744-1750.

14. L. Uhr, "Layered 'recognition cone' networks that preprocess, classify and describe," IEEE Trans. Computers 21, 1972, 758-768.

15. C. R. Dyer, "A quadtree machine for parallel image processing," Tech. Rept. KSL 51, University of Illinois at Chicago Circle, 1981.

16. S. Tanimoto, "Towards hierarchical cellular logic: design considerations for pyramid machines," Department of Computer Science Technical Report 81-02-01, University of Washington, 1981.

17. A. R. Hanson and E. M. Riseman, "Processing cones: a computational structure for image analysis," in S. Tanimoto and A. Klinger, eds., Structured Computer Vision, Academic Press, New York, 1980.

18. P. Burt, T. H. Hong, and A. Rosenfeld, "Segmentation and estimation of image region properties through cooperative hierarchical computation," IEEE Trans. Systems, Man, Cybernetics 11, 1981, 802-809.

19. R. Klette, "Log diameter restricted bottom-up triangle cellular acceptors (UTCA's)," TR-809, University of Maryland, College Park, MD, 1980.

20. N. Ahuja and S. Swamy, "Multiprocessor pyramids for bottom-up image analysis," Proc. PRIP, 1982, 380-385.

21. W. M. Newman and R. F. Sproull, "Principles of Interactive Computer Graphics," McGraw-Hill, 1979.

22. J. D. Foley and A. Van Dam, "Fundamentals of Interactive Computer Graphics," Addison-Wesley, 1980.

23. The TTL Data Book, Texas Instruments Incorporated, Dallas, TX, 1976.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR-TR- 83-0065	2. GOVT ACCESSION NO. AD-A125586	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) TOP-DOWN CELLULAR PYRAMIDS		5. TYPE OF REPORT & PERIOD COVERED Technical
		6. PERFORMING ORG. REPORT NUMBER TR-1191
7. AUTHOR(s) Angela Y. Wu Azriel Rosenfeld		8. CONTRACT OR GRANT NUMBER(s) AFOSR-77-3271
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Vision Laboratory Computer Science Center University of Maryland College Park, MD 20742		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61102F 2304/A2
11. CONTROLLING OFFICE NAME AND ADDRESS Math & Info. Sciences, AFOSR/NM Bolling AFB Washington, DC 20332		12. REPORT DATE July 1982 13. NUMBER OF PAGES 18
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Image processing
Computer graphics
Parallel processing
Cellular arrays
Pyramids

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A cellular pyramid is an exponentially tapering stack of arrays of processors ("cells"), where each cell is connected to its neighbors ("siblings") on its own level, to a "parent" on the level above, and to its "children" on the level below. It is shown that in some situations, if information flows top-down only, from fathers to sons, then a cellar pyrmaid may be no faster than a one-level cellular array; but it may be possible to use simpler cells in the pyramid case.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)


4-8
DTI